HELLO

This notebook introduces basic objects that populate a Smalltalk image.
In particular, we write in the Pharo dialect, which adhers to the following principles:

Array

```
1   'Easy to understand, easy to learn from, easy to change.'
2   'Objects all the way down.'
3   'Examples to learn from.'
4   'Fully dynamic and malleable.'
5   'Beauty in the code, beauty in the comments.'
6   'Simplicity is the ultimate elegance.'
7   'Better a set of small polymorphic classes than a large ugly one.'
8   'Classes structure our vocabulary.'
9   'Messages are our vocabulary.'
10  'Polymorphism is our esperanto.'
11  'Abstraction and composition are our friends.'
12  'Tests are important but can be changed.'
13  'Explicit is better than implicit.'
14  'Magic only at the right place.'
15  'One step at a time.'
16  'There is no unimportant fix.'
17  'Learning from mistakes.'
18  'Perfection can kill movement.'
19  'Quality is an emerging property.'
20  'Simple processes to support progress.'
21  'Communication is key.'
22  'A system with robust abstractions that a single person can understand.'
```

ProfStef pharoZenValues

For instance, let's create a very simple object,

```
o := Object new
```

Object

```
an Object
```

denoted by the symbol o. Wait for a moment and look at the little labels above each boxes.
They shows the *class* of the object represented in the box, so let's inspect it

```
o class
```

Object class

name Answer the name of the receiver.	#Object
class Primitive. Answer the object which is the receiver's class. Essential. See Object documentation whatIsAPrimitive.	
superclass Answer the receiver's superclass, a Class.	
methodsCount Answer the number of methods of the receiver that are currently installed in the `methodDictionary`.	531
instanceCount Answer the number of instances of the receiver that are currently in use. Note: we do not use #allInstances as allInstancesDo: has a fallback for low memory	197
category Answer the system organization category for the receiver.	#'Kernel-Objects'

asFluidDefinitionShape I produce a Bloaseal Pango text with the fluid definition of myself, according to `FluidClassDefinitionPrinter`.	`ProtoObject << #Object`
	`layout: FixedLayout;`
	`traits: {};`
	`slots: {};`
	`sharedVariables: {};`
	`sharedPools: {};`
	`tag: 'Objects';`
	`package: 'Kernel'`

Metaclass

name Answer a String that is the name of the receiver, either 'Metaclass' or the name of the receiver's class followed by ' class'.	'Object class'
class Primitive. Answer the object which is the receiver's class. Essential. See Object documentation whatIsAPrimitive.	
superclass Answer the receiver's superclass, a Class.	
methodsCount Answer the number of methods of the receiver that are currently installed in the `methodDictionary`.	27
instanceCount Answer the number of instances of the receiver that are currently in use. Note: we do not use #allInstances as allInstancesDo: has a fallback for low memory	1
category Answer the system organization category for the receiver.	#'Kernel-Objects'

asFluidDefinitionShape I produce a Bloaseal Pango text with the fluid definition of myself, according to `FluidClassDefinitionPrinter`.	`ProtoObject class << Object class`
	`traits: {};`
	`slots: {}`

ProtoObject class

name Answer the name of the receiver.	#ProtoObject
class Primitive. Answer the object which is the receiver's class. Essential. See Object documentation whatIsAPrimitive.	
superclass Answer the receiver's superclass, a Class.	nil
methodsCount Answer the number of methods of the receiver that are currently installed in the `methodDictionary`.	54
instanceCount Answer the number of instances of the receiver that are currently in use. Note: we do not use #allInstances as allInstancesDo: has a fallback for low memory	0
category Answer the system organization category for the receiver.	#'Kernel-Objects'

asFluidDefinitionShape I produce a Bloaseal Pango text with the fluid definition of myself, according to `FluidClassDefinitionPrinter`.	`nil << #ProtoObject`
	`layout: FixedLayout;`
	`traits: {};`
	`slots: {};`
	`sharedVariables: {};`
	`sharedPools: {};`
	`tag: 'Objects';`
	`package: 'Kernel'`

Metaclass class

name Answer the name of the receiver.	#Metaclass
class Primitive. Answer the object which is the receiver's class. Essential. See Object documentation whatIsAPrimitive.	
superclass Answer the receiver's superclass, a Class.	
methodsCount Answer the number of methods of the receiver that are currently installed in the `methodDictionary`.	80
instanceCount Answer the number of instances of the receiver that are currently in use. Note: we do not use #allInstances as allInstancesDo: has a fallback for low memory	9780
category Answer the system organization category for the receiver.	#'Kernel-CodeModel-Classes'

asFluidDefinitionShape I produce a Bloaseal Pango text with the fluid definition of myself, according to `FluidClassDefinitionPrinter`.	`ClassDescription << #Metaclass`
	`layout: FixedLayout;`
	`traits: {};`
	`slots: { #thisClass };`
	`sharedVariables: {};`
	`sharedPools: {};`
	`tag: 'Classes';`
	`package: 'Kernel-CodeModel'`

Metaclass

name Answer a String that is the name of the receiver, either 'Metaclass' or the name of the receiver's class followed by ' class'.	'ProtoObject class'
class Primitive. Answer the object which is the receiver's class. Essential. See Object documentation whatIsAPrimitive.	
superclass Answer the receiver's superclass, a Class.	
methodsCount Answer the number of methods of the receiver that are currently installed in the `methodDictionary`.	1
instanceCount Answer the number of instances of the receiver that are currently in use. Note: we do not use #allInstances as allInstancesDo: has a fallback for low memory	1
category	#'Kernel-Objects'

asFluidDefinitionShape I produce a Bloaseal Pango text with the fluid definition of myself, according to `FluidClassDefinitionPrinter`.	`Class class << ProtoObject class`
	`traits: {};`
	`slots: {}`

Metaclass

name Answer a String that is the name of the receiver, either 'Metaclass' or the name of the receiver's class followed by ' class'.	'Metaclass class'
class Primitive. Answer the object which is the receiver's class. Essential. See Object documentation whatIsAPrimitive.	
superclass Answer the receiver's superclass, a Class.	
methodsCount Answer the number of methods of the receiver that are currently installed in the `methodDictionary`.	1
instanceCount Answer the number of instances of the receiver that are currently in use. Note: we do not use #allInstances as allInstancesDo: has a fallback for low memory	1
category	#'Kernel-CodeModel-Classes'

asFluidDefinitionShape I produce a Bloaseal Pango text with the fluid definition of myself, according to `FluidClassDefinitionPrinter`.	`ClassDescription class << Metaclass class`
	`traits: {};`
	`slots: {}`

Metaclass

name Answer a String that is the name of the receiver, either 'Metaclass' or the name of the receiver's class followed by ' class'.	'ClassDescription class'
class Primitive. Answer the object which is the receiver's class. Essential. See Object documentation whatIsAPrimitive.	
superclass Answer the receiver's superclass, a Class.	
methodsCount Answer the number of methods of the receiver that are currently installed in the `methodDictionary`.	1
instanceCount Answer the number of instances of the receiver that are currently in use. Note: we do not use #allInstances as allInstancesDo: has a fallback for low memory	1
category	#'Kernel-CodeModel-Classes'

asFluidDefinitionShape I produce a Bloaseal Pango text with the fluid definition of myself, according to `FluidClassDefinitionPrinter`.	`Behavior class << ClassDescription class`
	`traits: {};`
	`slots: {}`

Metaclass

name Answer a String that is the name of the receiver, either 'Metaclass' or the name of the receiver's class followed by ' class'.	'Behavior class'
class Primitive. Answer the object which is the receiver's class. Essential. See Object documentation whatIsAPrimitive.	
superclass Answer the receiver's superclass, a Class.	
methodsCount Answer the number of methods of the receiver that are currently installed in the `methodDictionary`.	4
instanceCount Answer the number of instances of the receiver that are currently in use. Note: we do not use #allInstances as allInstancesDo: has a fallback for low memory	1
category	#'Kernel-CodeModel-Classes'

asFluidDefinitionShape I produce a Bloaseal Pango text with the fluid definition of myself, according to `FluidClassDefinitionPrinter`.	`Object class << Behavior class`
	`traits: {};`
	`slots: {}`

ClassDescription class

name Answer the name of the receiver.	#ClassDescription
class Primitive. Answer the object which is the receiver's class. Essential. See Object documentation whatIsAPrimitive.	
superclass Answer the receiver's superclass, a Class.	
methodsCount Answer the number of methods of the receiver that are currently installed in the `methodDictionary`.	228
instanceCount Answer the number of instances of the receiver that are currently in use. Note: we do not use #allInstances as allInstancesDo: has a fallback for low memory	0
category Answer the system organization category for the receiver.	#'Kernel-CodeModel-Classes'

asFluidDefinitionShape I produce a Bloaseal Pango text with the fluid definition of myself, according to `FluidClassDefinitionPrinter`.	`Behavior << #ClassDescription`
	`layout: FixedLayout;`
	`traits: {};`
	`slots: { #protocols };`
	`sharedVariables: {};`
	`sharedPools: {};`
	`tag: 'Classes';`
	`package: 'Kernel-CodeModel'`

Behavior class

name Answer the name of the receiver.	#Behavior
class Primitive. Answer the object which is the receiver's class. Essential. See Object documentation whatIsAPrimitive.	
superclass Answer the receiver's superclass, a Class.	
methodsCount Answer the number of methods of the receiver that are currently installed in the `methodDictionary`.	278
instanceCount Answer the number of instances of the receiver that are currently in use. Note: we do not use #allInstances as allInstancesDo: has a fallback for low memory	0
category Answer the system organization category for the receiver.	#'Kernel-CodeModel-Classes'

asFluidDefinitionShape I produce a Bloaseal Pango text with the fluid definition of myself, according to `FluidClassDefinitionPrinter`.	`Object << #Behavior`
	`layout: FixedLayout;`
	`traits: {};`
	`slots: { #superclass . #methodDict . #format . #layout };`
	`sharedVariables: { #ClassProperties . #ObsoleteSubclasses };`
	`sharedPools: {};`
	`tag: 'Classes';`
	`package: 'Kernel-CodeModel'`

Class class

name Answer the name of the receiver.	#Class
class Primitive. Answer the object which is the receiver's class. Essential. See Object documentation whatIsAPrimitive.	
superclass Answer the receiver's superclass, a Class.	
methodsCount Answer the number of methods of the receiver that are currently installed in the `methodDictionary`.	242
instanceCount Answer the number of instances of the receiver that are currently in use. Note: we do not use #allInstances as allInstancesDo: has a fallback for low memory	0
category Answer the system organization category for the receiver.	#'Kernel-CodeModel-Classes'

asFluidDefinitionShape I produce a Bloaseal Pango text with the fluid definition of myself, according to `FluidClassDefinitionPrinter`.	`ClassDescription << #Class`
	`layout: FixedLayout;`
	`traits: {};`
	`slots: { #subclasses . #name . #classPool . #sharedPools . #environment . #commentSourcePointer . #packageTag };`
	`sharedVariables: { #ObsoleteClasses };`
	`sharedPools: {};`
	`tag: 'Classes';`
	`package: 'Kernel-CodeModel'`

Metaclass

name Answer a String that is the name of the receiver, either 'Metaclass' or the name of the receiver's class followed by ' class'.	'Class class'
class Primitive. Answer the object which is the receiver's class. Essential. See Object documentation whatIsAPrimitive.	
superclass Answer the receiver's superclass, a Class.	
methodsCount Answer the number of methods of the receiver that are currently installed in the `methodDictionary`.	8
instanceCount Answer the number of instances of the receiver that are currently in use. Note: we do not use #allInstances as allInstancesDo: has a fallback for low memory	1
category	#'Kernel-CodeModel-Classes'

asFluidDefinitionShape I produce a Bloaseal Pango text with the fluid definition of myself, according to `FluidClassDefinitionPrinter`.	`ClassDescription class << Class class`
	`traits: {};`
	`slots: {}`

that is what's called the *Smalltalk object model*. Moreover, observe how such a class has been obtained, through the message `class` defined as follows:

```
ProtoObject >> #class
```

CompiledMethod

ProtoObject, protocol *class membership*.

Primitive. Answer the object which is the receiver's class. Essential. See Object documentation whatIsAPrimitive.

```
class

    <reflection: 'Object Inspection - Accessing object class'>
    <primitive: 111>
    self primitiveFailed
```

And in turn, have a look at what a `primitive` is:

```
#whatIsAPrimitive implementors
```

OrderedCollection

array	
firstIndex	1
lastIndex	1

Array

Object class, protocol *documentation*.

Some messages in the system are responded to primitively. A primitive response is performed directly by the interpreter rather than by evaluating expressions in a method. The methods for these messages indicate the presence of a primitive response by including <primitive: xx> or <primitive: 79 error: ec>before the first expression in the method.

In the second case, ec is a special temporary variable that holds an error code. In the VM primitive, failure is indicated by a variable called primFailCode being non-zero. On activating a method with a failing primitive, if the index is in bounds of the primitiveFailCodes array in the VMspecialObjectsArray then the failure code substitutes the symbol in the primitiveFailCodes array, otherwise it supplies the integer value.

Primitives exist for several reasons. Certain basic or 'primitive' operations cannot be performed in any other way. Smalltalk without primitives can move values from one variable to another, but cannot add two SmallIntegers together. Many methods for arithmetic and comparison between numbers are primitives. Some primitives allow Smalltalk to communicate with I/O devices such as the disk, the display, and the keyboard. Some primitives exist only to make the system run faster; each does the same thing as a certain Smalltalk method, and its implementation as a primitive is optional.

When the Smalltalk interpreter begins to execute a method which specifies a primitive response, it tries to perform the primitive action and to return a result. If the routine in the interpreter for this primitive is successful, it will return a value and the expressions in the method will not be evaluated. If the primitive routine is not successful, the primitive 'fails', and the Smalltalk expressions in the method are executed instead. These expressions are evaluated as though the primitive routine had not been called.

1 The Smalltalk code that is evaluated when a primitive fails usually anticipates why that primitive might fail. If the primitive is optional, the expressions in the method do exactly what the primitive would have done (See Number @). If the primitive only works on certain classes of arguments, the Smalltalk code tries to coerce the argument or appeals to a superclass to find a more general way of doing the operation (see SmallInteger +). If the primitive is never supposed to fail, the expressions signal an error (see SmallInteger asFloat).

Each method that specifies a primitive has a comment in it. If the primitive is optional, the comment will say 'Optional'. An optional primitive that is not implemented always fails, and the Smalltalk expressions do the work instead.

If a primitive is not optional, the comment will say, 'Essential'. Some methods will have the comment, 'No Lookup'. See Object howToModifyPrimitives for an explanation of special selectors which are not looked up.

For the primitives for +, -, *, and bitShift: in SmallInteger, and truncated in Float, the primitive constructs and returns a 16-bit LargePositiveInteger when the result warrants it. Returning 16-bit LargePositiveIntegers from these primitives instead of failing is optional in the same sense that the LargePositiveInteger arithmetic primitives are optional. The comments in the SmallInteger primitives say, 'Fails if result is not a SmallInteger', even though the implementor has the option to construct a LargePositiveInteger. For further information on primitives, see the 'Primitive Methods' part of the chapter on the formal specification of the interpreter in the Smalltalk book.

```
whatIsAPrimitive

    self error: 'comment only'
```

2	nil
3	nil
4	nil
5	nil
6	nil
7	nil
8	nil
9	nil
10	nil

Also observe that o has been created by the message

```
o class class lookupSelector: #new
```

CompiledMethod

Behavior, protocol *instance creation*.

Answer a new initialized instance of the receiver (which is a class) with no indexable variables. Fail if the class is indexable.

```
new

    ^ self basicNew initialize
```